PYTHON BASIC SYNTAX

The Python language has many similarities to Perl, C, and Java. However, there are some definite differences between the languages. This chapter is designed to quickly get you up to speed on the syntax that is expected in Python.

First Python Program:

Interactive Mode Programming:

Invoking the interpreter without passing a script file as a parameter brings up the following prompt:

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type the following text to the right of the Python prompt and press the Enter key:

```
>>> print "Hello, Python!";
```

If you are running new version of Python then you would need to use print statement with parenthesis like **print ("Hello, Python!");**. However at Python version 2.4.3, this will produce following result:

```
Hello, Python!
```

Script Mode Programming :

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. All python files will have extension **.py**. So put the following source code in a test.py file.

```
print "Hello, Python!";
```

Here I assumed that you have Python interpreter set in PATH variable. Now try to run this program as follows:

```
$ python test.py
```

This will produce following result:

```
Hello, Python!
```

Let's try another way to execute a Python script. Below is the modified test.py file:

```
#!/usr/bin/python

print "Hello, Python!";
```

Here I assumed that you have Python interpreter available in /usr/bin directory. Now try to run this program as follows:

```
$ chmod +x test.py     # This is to make file executable
```

```
$./test.py
```

This will produce following result:

```
Hello, Python!
```

Python Identifiers:

A Python identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus **Manpower** and **manpower** are two different identifiers in Python.

Here are following identifier naming convention for Python:

- Class names start with an uppercase letter and all other identifiers with a lowercase letter.
- Starting an identifier with a single leading underscore indicates by convention that the identifier is meant to be private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Reserved Words:

The following list shows the reserved words in Python. These reserved words may not be used as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Lines and Indentation:

One of the first caveats programmers encounter when learning Python is the fact that there are no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. Both blocks in this example are fine:

```
if True:
    print "True"
else:
  print "False"
```

However, the second block in this example will generate an error:

```
if True:
    print "Answer"
    print "True"
else:
    print "Answer"
  print "False"
```

Thus, in Python all the continous lines indented with similar number of spaces would form a block. Following is the example having various statement blocks:

Note: Don't try to understand logic or different functions used. Just make sure you undertood various blocks even if they are without braces.

```
#!/usr/bin/python

import sys

try:
  # open file stream
  file = open(file_name, "w")
except IOError:
  print "There was an error writing to", file_name
  sys.exit()
print "Enter '", file_finish,
print "' When finished"
while file_text != file_finish:
  file_text = raw_input("Enter text: ")
  if file_text == file_finish:
    # close the file
    file.close
    break
  file.write(file_text)
  file.write("\n")
file.close()
file_name = raw_input("Enter filename: ")
if len(file_name) == 0:
  print "Next time please enter something"
  sys.exit()
try:
  file = open(file_name, "r")
except IOError:
  print "There was an error reading file"
  sys.exit()
file_text = file.read()
file.close()
print file_text
```

Multi-Line Statements:

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example:

```
total = item_one + \
        item_two + \
        item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example:

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation in Python:

Python accepts single ('), double (") and triple ('' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes can be used to span the string across multiple lines. For example, all the following are legal:

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python:

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the physical line end are part of the comment, and the Python interpreter ignores them.

```
#!/usr/bin/python

# First comment
print "Hello, Python!";  # second comment
```

This will produce following result:

```
Hello, Python!
```

A comment may be on the same line after a statement or expression:

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Using Blank Lines:

A line containing only whitespace, possibly with a comment, is known as a blank line, and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Waiting for the User:

The following line of the program displays the prompt, Press the enter key to exit. and waits for the user to press the Enter key:

```
#!/usr/bin/python

raw_input("\n\nPress the enter key to exit.")
```

Here "\n\n" are being used to create two new lines before displaying the actual line. Once the user presses the key, the

program ends. This is a nice trick to keep a console window open until the user is done with an application.

Multiple Statements on a Single Line:

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Multiple Statement Groups as Suites:

A groups of individual statements which makes a single code block are called **suites** in Python. Compound or complex statements, such as if, while, def, and class, are those which require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite. For example:

```
if expression :
   suite
elif expression :
   suite
else :
   suite
```

Command Line Arguments:

You may have seen, for instance, that many programs can be run so that they provide you with some basic information about how they should be run. Python enables you to do this with -h:

```
$ python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser (also PYTHONDEBUG=x)
-E     : ignore environment variables (such as PYTHONPATH)
-h     : print this help message and exit

[ etc. ]
```

You can also program your script in such a way that it should accept various options. Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.